```
#Jack Amos
#CS2300-002
#11/22/2019
#Project 4
#Python 3.7

#starting data
years = [1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0]
income = [5000.0,7500.0,15000.0,20000.0,66000.0,72000.0,74500.0,80000.0,82000.0,98000.0]

#checks system has proper number of 0's
def sysCheck(sys):

    num_zeros = 0
    count = len(sys)-1

    while count != 0:
        num_zeros+=count
        count-=1


    #sets completed rows corresponding index to 0
    zero_count = 0


    for n in sys:
        for m in n:
            if m == 0.0:
                zero_count+=1

    if zero_count < num_zeros:

        return False

    return True



def solveGaussianSystem(degree):

    #get degree+1 so it forms system of proper dimensions
    dim = degree+1

    #create array of values for guassian system
    system = []
    xSigma = 0.0
    power = 0
    iterator = dim
```

```
while len(system) < (dim*dim):

    for n in years:

        xSigma+=pow(n,power)
    power+=1

    system.append(xSigma)
    xSigma = 0.0

    if power == iterator:

        power-=1
        iterator+=degree


#create gaussian product arrays
prod_values = []
ySigma = 0.0
power = 0
i = 0

while len(prod_values) < dim:

    while i < len(years):

        ySigma+=(income[i]*pow(years[i],power))
        i+=1

    power+=1
    i = 0

    prod_values.append(ySigma)
    ySigma = 0.0


#making arrays of points into gaussian system
count = dim*dim
iterator = dim
start = 0
end = dim
temp = []
sysFinal = []

while count != 0:

    temp = system[start:end]
    sysFinal.append(temp)
    temp = []
```

```
        count-=dim
        start+=iterator
        end+=iterator


    #make necessary 0's
    check = False
    col_round = 1
    top = 0
    top_value = 0.0
    current = 0
    current_value = 0.0


    while check == False:

        #gaussian solving logic
        top_value = sysFinal[top][top]
        current = top + 1

        while current < len(sysFinal):

            current_value = sysFinal[current][top]

            if current_value != 0.0:

                row_length = 0
                prod_values[current] = (prod_values[current] * top_value * -1) + (prod_values[top] * current_value)

                while row_length < len(sysFinal):

                    sysFinal[current][row_length] = (sysFinal[current][row_length] * top_value * -1) + (sysFinal[top][row_length] * current_value)

                    row_length+=1


            current+=1

        top+=1


        #set value to 0 if below or at minimum of .0000001, -.0000001, -0, and keeps values from becoming too small
        for n in sysFinal:
            for m in n:

                if m <= .0000001 and m > 0.0:
                    n[n.index(m)] = 0.0
                elif m >= -.0000001 and m < 0.0:
                    n[n.index(m)] = 0.0
                elif m == -0.0:
                    n[n.index(m)] = 0.0

        for n in prod_values:

            if n <= .0000001 and n > 0.0:
                n = 0.0
            elif n >= -.0000001 and n < 0.0:
                n = 0.0
            elif n == -0.0:
                n = 0.0


        #reduces size of floating point values without removing them
        big_count = 0
        current_array = []

        for n in sysFinal:

            current_array = sysFinal[sysFinal.index(n)]

            if prod_values[sysFinal.index(n)] > 100 or prod_values[sysFinal.index(n)] < -
100:

                prod_values[sysFinal.index(n)] = prod_values[sysFinal.index(n)]/100

                while big_count < len(sysFinal):

                    current_array[big_count] = current_array[big_count]/100

                    big_count+=1

            big_count = 0


        col_round+=1

        check = sysCheck(sysFinal)

    #solve for x values, starting at bottom
    x_values = []
    i = 0

    #make array of x's based on matrix size
    while i < len(sysFinal):
        x_values.append(1)
        i+=1

    i-=1
    solveX = 0.0
    f = len(sysFinal)-1
    current_array = sysFinal[len(sysFinal)-1]
    divVal = 0.0

    #gets sum of all current x's and row values then subtracts non important x positions and divides product value by important x
    while f > -1:

        while i > -1:

            if i != f:
                solveX += x_values[i] * current_array[i]
            elif i == f:
                divVal = current_array[i]
            i-=1

        x_values[f] = (prod_values[f] - solveX)/divVal
        f-=1
        i = len(sysFinal)-1
        solveX = 0.0
        current_array = sysFinal[f]


    #create equation string
    equation_string = ""
    i = 2

    equation_string = str("{:.4f}".format(x_values[1]))+"x+"+str("{:.4f}".format(x_values[0]))

    while i < len(x_values):

        equation_string = str("{:.4f}".format(x_values[i]))+"x^"+str(i)+"+"+equation_string
        i+=1


    if degree == 1:
        equation_string = "Linear: "+equation_string
    elif degree == 2:
        equation_string = "Quadratic: "+equation_string

    #find income at year 15
    power = 0
    answer = 0.0
    for n in x_values:

        answer+=pow(15,power)*n
        power+=1


    equation_string+="     Income at year 15: $"+str("{:.2f}".format(answer))
    equation_string = "\n"+equation_string

    #print coefficients
    print(equation_string)


#func calls
solveGaussianSystem(1)
solveGaussianSystem(2)
```